Aperçu du protocole QUIC – Document Expérimental

Flavien Haas, Louis Royer, Daniel Sanchez Pelegrin, Maïmouna Diouf, et Carine Koumsaga

UPSSITECH, Université Paul Sabatier, Toulouse, France

Février 2020

Résumé. QUIC est un protocole de transport actuellement au stade de *draft* dans le processus de standardisation par l'IETF. Son but principal est de réduire le nombre d'aller-retours en incluant des informations de chiffrement et des données dès les premiers échanges. QUIC s'implémente au-dessus d'UDP, ce qui lui permet une flexibilité et une évolutivité.
Ce document montre le résultat de nos expérimentations sur les différentes implémentations du protocole QUIC que nous avons pu tester correspondant aux *drafts* actuels.

Sommaire

1 Introduction

Lors de nos recherches, nous avons voulu montrer expérimentalement comment se passent les échanges lors de l'accès à une page web. Le scénario classique permettant de mettre en évidence les avantages de QUIC par rapport à *HTTP over TLS+TCP* est le cas d'un client qui se connecte plusieurs fois à un site web et avec des connexions en parallèle ; le but de QUIC est alors d'obtenir plus vite le contenu en réduisant les allers-retours, tout en atteignant les objectifs de sécurité (confidentialité, intégrité, authentification).

On veut donc voir un échange en *0-RTT* pour pouvoir le comparer avec *TLS+TCP*. On voudrait également avoir des métriques comme le temps total avant que l'utilisateur puisse obtenir sa page.

Plusieurs implémentations respectant les *drafts* du protocole QUIC sont disponibles[1] :

- ngtcp2 (librairie avec un client et serveur)
- Kwik (seulement un client)
- Neqo
- Quant (librairie avec client et serveur mais pas HTTP/3)
- Quiche et le module NGINX basé sur quiche
- Curl (seulement un client, support expérimental)

1 https://github.com/quicwg/base-drafts/wiki/Implementations

2 Méthodologie

Les serveurs/clients disponibles ne sont pas tous compatibles entre eux puisqu'ils n'implémentent pas encore tous complètement les drafts. Nous avons effectué la majorité de nos tests avec ngtcp2[2].

Nous avons essayé de compiler plusieurs implémentations, dont Quiche, Neqo, et le support expérimental de HTTP/3 dans Curl[3] mais nous obtenons des erreurs lors de la compilation lors de l'étape d'édition des liens[4]. Dans tous ces cas, il faut utiliser une version patchée de TLS que nous n'avons réussi à faire fonctionner QUIC correctement qu'avec ngtcp2.

Nous avons également réussi à faire fonctionner quant, mais il n'est pas interopérable avec ngtcp2 puisqu'il n'implémente pas HTTP/3, mais un HTTP/0.9.

Pour nos exemples, nous avons généré un certificat TLS1.3 grâce à la version patchée d'OpenSSL.

On observe le trafic avec Wireshark, mais les données de QUIC étant chiffrées, il faut récupérer les clés de l'échange (ce que permet de faire ngtcp2) et les injecter dans Wireshark. Il faut au minimum la version 3.2.2 de Wireshark pour avoir le support du *draft 25*[5] de QUIC.

2 commit ngtcp2 : 82a5b8cc26dd2cf4270446422212345bc2dfe507
3 Curl : https://github.com/curl/curl/wiki/HTTP3
4 Doc Curl : https://github.com/curl/curl/blob/master/docs/HTTP3.md
5 Draft 25 : https://github.com/quicwg/base-drafts/wiki/Tools#wireshark

3 Résultats/Mesures

3.1 ngtcp2

Notre environnement de test est composé de deux conteneurs LXC (un pour le serveur et un pour le client) dont l'image est basée sur debian 10.3. La version utilisée pour ce test est le commit 82a5b8cc26dd2cf4270446422212345bc2dfe507.

Une fois les librairies nghttp3 et OpenSSL patchée compilée, nous pouvons compiler les exemples client/serveur de ngtcp2.

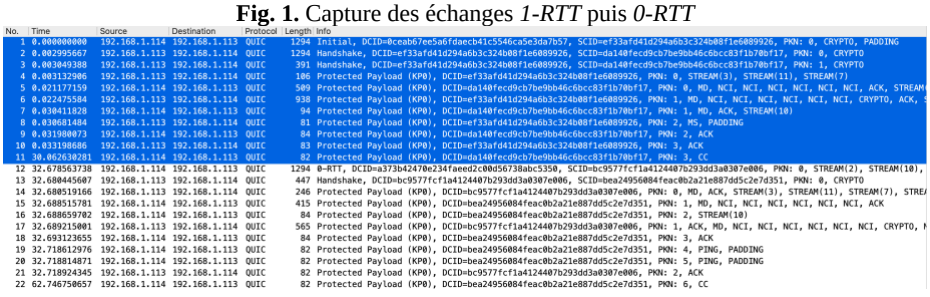
L'option nous permettant de mettre en œuvre le *0-RTT* pour le serveur est `--early-response`, pour activer le mode early data. Pour une liste complète des options, on peut utiliser l'option `--help`.

```
root@quicServer:~/ngtcp2/examples# ./server --early-response 0.0.0.0 11380
/root/certificates/ca.key /root/certificates/ca.cer
```

Pour reprendre une session, un ticket de session et un paramètre de transport du serveur doivent être enregistrés par le client avec les options `--session-file` et `--tp-file`. On définit la variable d'environnement `SSLKEYLOGFILE` qui contient un chemin vers l'emplacement où écrire le secret partagé qu'il faudra importer dans Wireshark.

```
root@quicClient:~/ngtcp2/examples# SSLKEYLOGFILE=ngtcp2.keys ./client  --
session-file="/root/ngtcp2/examples/sessionfile" --tp-
file="/root/ngtcp2/examples/tpfile" 192.168.1.113 11380
https://192.168.1.113/index.html
```

Fig. 1. Capture des échanges *1-RTT* puis *0-RTT*



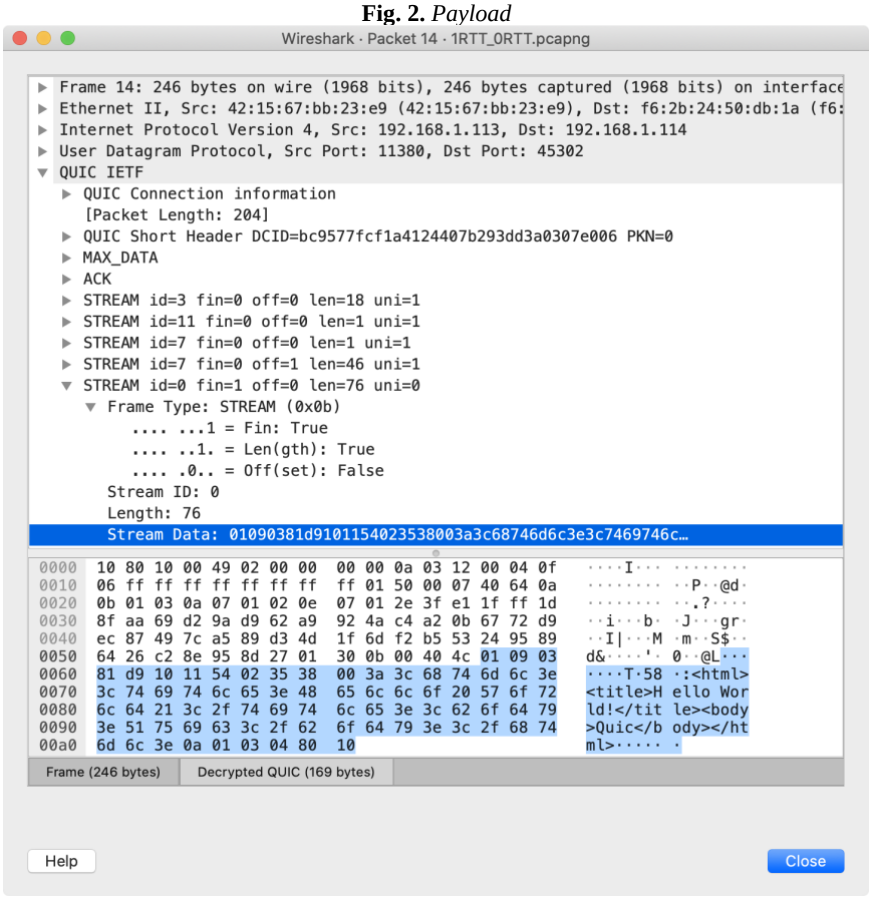
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.1.114	192.168.1.113	QUIC	1294	Initial, DCID=0ceab67ee5a6fdaecb41c5546ca5e3da7b57, SCID=ef33afd41d294a6b3c324b08f1e6089926, PKN: 0, CRYPTO, PADDING
2	0.002995667	192.168.1.113	192.168.1.114	QUIC	1294	Handshake, DCID=ef33afd41d294a6b3c324b08f1e6089926, SCID=da140fecd9cb7be9bb46c6bcc83f1b70bf17, PKN: 0, CRYPTO
3	0.003049388	192.168.1.113	192.168.1.114	QUIC	391	Handshake, DCID=ef33afd41d294a6b3c324b08f1e6089926, SCID=da140fecd9cb7be9bb46c6bcc83f1b70bf17, PKN: 1, CRYPTO
4	0.003132906	192.168.1.113	192.168.1.114	QUIC	106	Protected Payload (KP0), DCID=ef33afd41d294a6b3c324b08f1e6089926, PKN: 0, STREAM(3), STREAM(11), STREAM(7)
5	0.021177159	192.168.1.114	192.168.1.113	QUIC	589	Protected Payload (KP0), DCID=da140fecd9cb7be9bb46c6bcc83f1b70bf17, PKN: 0, MD, NCI, NCI, NCI, NCI, NCI, NCI, ACK, STREAM(
6	0.022475584	192.168.1.113	192.168.1.114	QUIC	938	Protected Payload (KP0), DCID=ef33afd41d294a6b3c324b08f1e6089926, PKN: 1, MD, NCI, NCI, NCI, NCI, NCI, NCI, CRYPTO, ACK, S
7	0.030411828	192.168.1.114	192.168.1.113	QUIC	94	Protected Payload (KP0), DCID=da140fecd9cb7be9bb46c6bcc83f1b70bf17, PKN: 1, MD, ACK, STREAM(10)
8	0.030681484	192.168.1.113	192.168.1.114	QUIC	81	Protected Payload (KP0), DCID=ef33afd41d294a6b3c324b08f1e6089926, PKN: 2, MS, PADDING
9	0.031980073	192.168.1.114	192.168.1.113	QUIC	84	Protected Payload (KP0), DCID=da140fecd9cb7be9bb46c6bcc83f1b70bf17, PKN: 2, ACK
10	0.033198686	192.168.1.113	192.168.1.114	QUIC	83	Protected Payload (KP0), DCID=ef33afd41d294a6b3c324b08f1e6089926, PKN: 3, ACK
11	30.062630201	192.168.1.114	192.168.1.113	QUIC	82	Protected Payload (KP0), DCID=da140fecd9cb7be9bb46c6bcc83f1b70bf17, PKN: 3, CC
12	32.678563738	192.168.1.114	192.168.1.113	QUIC	1294	0-RTT, DCID=a373b42470e234faeed2c00d56738abc5350, SCID=bc9577fcf1a4124407b293dd3a0307e006, PKN: 0, STREAM(2), STREAM(10),
13	32.680445607	192.168.1.113	192.168.1.114	QUIC	447	Handshake, DCID=bc9577fcf1a4124407b293dd3a0307e006, SCID=bea24956084feac0b2a21e887dd5c2e7d351, PKN: 0, CRYPTO
14	32.680519166	192.168.1.113	192.168.1.114	QUIC	246	Protected Payload (KP0), DCID=bc9577fcf1a4124407b293dd3a0307e006, PKN: 0, MD, ACK, STREAM(3), STREAM(11), STREAM(7), STREA
15	32.688515781	192.168.1.114	192.168.1.113	QUIC	415	Protected Payload (KP0), DCID=bea24956084feac0b2a21e887dd5c2e7d351, PKN: 1, MD, NCI, NCI, NCI, NCI, NCI, NCI, ACK
16	32.688659702	192.168.1.114	192.168.1.113	QUIC	84	Protected Payload (KP0), DCID=bea24956084feac0b2a21e887dd5c2e7d351, PKN: 2, STREAM(10)
17	32.689215001	192.168.1.113	192.168.1.114	QUIC	565	Protected Payload (KP0), DCID=bc9577fcf1a4124407b293dd3a0307e006, PKN: 1, ACK, MD, NCI, NCI, NCI, NCI, NCI, NCI, CRYPTO, N
18	32.693123655	192.168.1.114	192.168.1.113	QUIC	84	Protected Payload (KP0), DCID=bea24956084feac0b2a21e887dd5c2e7d351, PKN: 3, ACK
19	32.718612976	192.168.1.114	192.168.1.113	QUIC	82	Protected Payload (KP0), DCID=bea24956084feac0b2a21e887dd5c2e7d351, PKN: 4, PING, PADDING
20	32.718814871	192.168.1.114	192.168.1.113	QUIC	82	Protected Payload (KP0), DCID=bea24956084feac0b2a21e887dd5c2e7d351, PKN: 5, PING, PADDING
21	32.718924345	192.168.1.113	192.168.1.114	QUIC	82	Protected Payload (KP0), DCID=bc9577fcf1a4124407b293dd3a0307e006, PKN: 2, ACK
22	62.746750657	192.168.1.114	192.168.1.113	QUIC	82	Protected Payload (KP0), DCID=bea24956084feac0b2a21e887dd5c2e7d351, PKN: 6, CC

Nous pouvons voir que lors de la première requête, étant donné que le client et le serveur ne se connaissent pas, la demande est effectuée avec un paquet de type *INITIAL*.

Lors de la seconde requête, le fichier de session contenant des paramètres de session TLS ainsi que le fichier de paramètre de transport existant pour ce serveur sont utilisés par le client pour effectuer sa requête en *0-RTT*.

La *payload* est contenue dans le 5ème paquet après la demande (*1-RTT*, paquet numéro 6) et 2ème après la demande pour le *0-RTT* (paquet numéro 14).

Fig. 2. *Payload*



Par défaut, le client peut demander au plus l'ouverture de 100 streams pour sa requête, mais le serveur n'en accepte que 3 au maximum. Lorsque nous avons voulu limiter à un seul stream en utilisant le paramètre `--max-streams-uni=1` pour le client et pour le serveur, cela n'a pas marché et nous obtenons l'erreur suivante :

Sur le client :

```
Initial(0x00) CONNECTION_CLOSE(0x1c) error_code=CRYPTO_ERROR(0x150)
frame_type=0 reason_len=0 reason=[]
ngtcp2_conn_read_pkt: ERR_DRAINING
ngtcp2_conn_write_connection_close: ERR_INVALID_STATE
```

Sur le serveur :

```
peer does not allow at least 3 unidirectional streams.
ngtcp2_conn_read_pkt: ERR_CRYPTO
Closing period has started (0.078 seconds)
```

3.2 quant

Notre environnement de test est composé d'une machine sous macOS 10.15.3 qui héberge le client et le serveur.

La version que nous avons utilisée pour réaliser ce test est le commit 67af12fff7c614ce5dab8676b160c25d0f2f56f7.

Initialisation du serveur :

```
~/quant/Debug/bin$  ./server -d /Users/flavien/quant/Debug/bin/servroot/ -c
/Users/flavien/certificates/ca.cer -k /Users/flavien/certificates/ca.key
0.038   q_init quic.c:657 quant/socket (kqueue/sendmsg/recvmsg) 0.0.27/67af12f
ready
        q_init quic.c:658 submit bug reports at
https://github.com/NTAP/quant/issues
0.066   q_bind quic.c:448 bound serv socket to [::1]:4433
        main server.c:332 server waiting on lo0 ::1:4433
        q_bind quic.c:448 bound serv socket to [fe80::1]:4433
        main server.c:332 server waiting on lo0 fe80::1:4433
        q_bind quic.c:448 bound serv socket to 127.0.0.1:4433
        main server.c:332 server waiting on lo0 127.0.0.1:4433
        q_bind quic.c:448 bound serv socket to [::1]:4434
        main server.c:332 server waiting on lo0 ::1:4434
        q_bind quic.c:448 bound serv socket to [fe80::1]:4434
        main server.c:332 server waiting on lo0 fe80::1:4434
        q_bind quic.c:448 bound serv socket to 127.0.0.1:4434
        main server.c:332 server waiting on lo0 127.0.0.1:4434
6.551   log_pkt pkt.c:109 RX from=[::1]:59284 len=1200
0xc3=Initialvers=0xbabababa dcid=0:17cefa9fc302c953a39df4299368
scid=0:537c3e71 tok=len=1172 nr=0
        rx_pkts conn.c:1318 clnt-requested vers 0xbabababa not supported
        tx_vneg_resp conn.c:349 sending vneg serv response
6.551   log_pkt pkt.c:138 TX to=[::1]:59284 0xaf=Version Negotiation vers=0x0
dcid=0:537c3e71 scid=0:17cefa9fc302c953a39df4299368
6.553   rx_pkts conn.c:1329 new serv conn on port 4434 from [::1]:38119
w/cid=0:af15542d90e3a9f5d14e9068be5f18
        new_conn conn.c:1988 other socket is 127.0.0.1:4434
        new_conn conn.c:2020 serv conn 0:af15542d90e3a9f5d14e9068be5f18 on
port 4434 created
6.553   log_pkt pkt.c:109 RX from=[::1]:59284 len=1184 0xc0=Initial
vers=0x4547471b dcid=0:af15542d90e3a9f5d14e9068be5f18 scid=0:7775ee1c tok=
len=1171 nr=0
        rx_pkt conn.c:1023 sending retry
        make_rtry_tok tls.c:1729 computed Retry tok
8393ca9cbbcc6b5d751eceefce35bd363887466023ccab95487f9de78a1a2473af15542d90e3a9
f5d14e9068be5f18
```

```
        update_act_scid conn.c:755 hshk switch to scid 0:59dbff15 for
conn_idle serv conn (was 0:af15542d90e3a9f5d14e9068be5f18)
6.554   log_pkt pkt.c:146 TX to=[::1]:59284 0xfe=Retry vers=0x4547471b
dcid=0:7775ee1c scid=0:59dbff15
tok=8393ca9cbbcc6b5d751eceefce35bd363887466023ccab95487f9de78a1a2473af15542d90
e3a9f5d14e9068be5f18 rit=77545599b9a8d707f25b5a3637ee56b7
        log_sent_pkts conn.c:292 serv Initial unacked: 0
        rx_pkts conn.c:1329 new serv conn on port 4434 from [::1]:38119
w/cid=0:59dbff15
        new_conn conn.c:1988 other socket is 127.0.0.1:4434
        new_conn conn.c:2020 serv conn 0:59dbff15 on port 4434 created
6.555   log_pkt pkt.c:109 RX from=[::1]:59284 len=1184 0xc0=Initial
vers=0x4547471b dcid=0:59dbff15 scid=0:7775ee1c
tok=8393ca9cbbcc6b5d751eceefce35bd363887466023ccab95487f9de78a1a2473af15542d90
e3a9f5d14e9068be5f18 len=1135 nr=0
        verify_rtry_tok tls.c:1751 computed Retry tok
8393ca9cbbcc6b5d751eceefce35bd363887466023ccab95487f9de78a1a2473
        log_pad frame.c:1146 PADDING len=26
        log_stream_or_crypto_frame frame.c:156 CRYPTO off=0 len=295 [seq]
        log_pad frame.c:1146 PADDING len=794
        update_act_scid conn.c:755 hshk switch to scid 0:003b3553 for
conn_opng serv conn (was 0:59dbff15)
        on_ch tls.c:348         SNI = localhost
        on_ch tls.c:378         ALPN = hq-27
        chk_tp tls.c:464         initial_max_stream_data_uni = 2047 [bytes]
        chk_tp tls.c:518         max_packet_size = 2000 [bytes]
        chk_tp tls.c:646         active_connection_id_limit = 4
        chk_tp tls.c:511         max_idle_timeout = 10000 [ms]
        chk_tp tls.c:483         initial_max_stream_data_bidi_remote = 65535
[bytes]
        chk_tp tls.c:446         private tp (0xff2d w/len 9) = 9044be24b004...
        chk_tp tls.c:473         initial_max_stream_data_bidi_local = 65535
[bytes]
        chk_tp tls.c:490         initial_max_data = 393210 [bytes]
        chk_tp tls.c:530         ack_delay_exponent = 3
        chk_tp tls.c:504         initial_max_streams_uni = 3
        chk_tp tls.c:543         max_ack_delay = 25 [ms]
        chk_tp tls.c:497         initial_max_streams_bidi = 6
6.556   err_close conn.c:1704 TLS error 515
6.556   log_pkt pkt.c:156 TX to=[::1]:59284 0xc0=Initial vers=0x4547471b
dcid=0:7775ee1c scid=0:003b3553 tok= len=0 nr=0
        enc_ack_frame frame.c:1534 ACK 0x02= lg=0 delay=0 (0 usec) cnt=0 rng=0
[0]
        enc_close_frame frame.c:1667 CONNECTION_CLOSE 0x1c=quic err=0x103
frame=0x06 rlen=14 reason=TLS error 515
```

```
        log_sent_pkts conn.c:292 serv Initial unacked: 0
8.055   q_close quic.c:731 closing serv conn 0:003b3553 on [::1]:4434 w/err
0x0
        q_close quic.c:802 serv conn 0:003b3553 stats:
        q_close quic.c:804 pkts_in_valid = 1
        q_close quic.c:806 pkts_in_invalid = 0
        q_close quic.c:807 pkts_out = 1
        q_close quic.c:808 pkts_out_lost = 0
        q_close quic.c:809 pkts_out_rtx = 0
        q_close quic.c:812 rtt = 0.000 (min = 0.000, max = 0.000, var = 0.000)
        q_close quic.c:814 cwnd = 12000 (max = 0)
        q_close quic.c:816 ssthresh = 0
        q_close quic.c:817 pto_cnt = 0
        q_close quic.c:818 frame                   code        out         in
        q_close quic.c:824 PADDING                 0x00          0        820
        q_close quic.c:824 ACK                     0x02          1          0
        q_close quic.c:824 CRYPTO                  0x06          0          1
        q_close quic.c:824 CONNECTION_CLOSE_QUIC   0x1c          1          0
        q_close quic.c:826 strm_frms_in_seq = 1
        q_close quic.c:827 strm_frms_in_ooo = 0
        q_close quic.c:828 strm_frms_in_dup = 0
        q_close quic.c:829 strm_frms_in_ign = 0
18.057   q_close quic.c:731 closing serv conn 0: on 127.0.0.1:4434 w/err 0x0
         q_close quic.c:731 closing serv conn 0: on [fe80::1]:4434 w/err 0x0
         q_close quic.c:731 closing serv conn 0: on [::1]:4434 w/err 0x0
         q_close quic.c:731 closing serv conn 0: on 127.0.0.1:4433 w/err 0x0
         q_close quic.c:731 closing serv conn 0: on [fe80::1]:4433 w/err 0x0
         q_close quic.c:731 closing serv conn 0: on [::1]:4433 w/err 0x0
18.068   main server.c:421 server exiting
```

Requete du client :

```
~/quant/Debug/bin$ ./client https://localhost:4434/index.html
0.032   q_init quic.c:657 quant/socket (kqueue/sendmsg/recvmsg) 0.0.27/67af12f
ready
        q_init quic.c:658 submit bug reports at
https://github.com/NTAP/quant/issues
0.055   read_tickets tls.c:1329 reading TLS tickets from /tmp/quant-session
        read_tickets tls.c:1334 could not read TLS tickets from /tmp/quant-
session
0.057   main client.c:519 client retrieving https://localhost:4434/index.html
0.063   new_conn conn.c:2020 clnt conn 0:537c3e71 on port 59284 created
        q_connect quic.c:272 new 1-RTT clnt conn 0:537c3e71 to [::1]:4434, 17
bytes queued for TX
0.064   q_connect quic.c:295 waiting for connect on clnt conn 0:537c3e71 to
[::1]:4434
```

```
0.064   log_pkt pkt.c:156 TX to=[::1]:4434 0xc0=Initial vers=0xbabababa
dcid=0:17cefa9fc302c953a39df4299368 scid=0:537c3e71 tok= len=0 nr=0
        enc_padding_frame frame.c:1450 PADDING len=15
        log_stream_or_crypto_frame frame.c:156 CRYPTO off=0 len=295 [seq]
        enc_padding_frame frame.c:1450 PADDING len=841
        log_cc recovery.c:143 clnt conn 0:537c3e71: in_flight=1200 (+1200),
cwnd=12000 (+0), ssthresh=0 (+0), srtt=0.000 (+0.000), rttvar=0.000 (+0.000)
        log_sent_pkts conn.c:292 clnt Initial unacked: 0
0.066   log_pkt pkt.c:90 RX from=[::1]:4434 len=33 0xaf=Version Negotiation
vers=0x0 dcid=0:537c3e71 scid=0:17cefa9fc302c953a39df4299368
        rx_pkt conn.c:1121 serv didn't like vers 0xbabababa, retrying with
0x4547471b
0.067   log_pkt pkt.c:156 TX to=[::1]:4434 0xc0=Initial vers=0x4547471b
dcid=0:af15542d90e3a9f5d14e9068be5f18 scid=0:7775ee1c tok= len=0 nr=0
        enc_padding_frame frame.c:1450 PADDING len=14
        log_stream_or_crypto_frame frame.c:156 CRYPTO off=0 len=295 [seq]
        enc_padding_frame frame.c:1450 PADDING len=841
        log_cc recovery.c:143 clnt conn 0:7775ee1c: in_flight=1200 (+1200),
cwnd=12000 (+0), ssthresh=0 (+0), srtt=0.000 (+0.000), rttvar=0.000 (+0.000)
        log_sent_pkts conn.c:292 clnt Initial unacked: 0
0.068   add_dcid conn.c:807 hshk switch to dcid 0:59dbff15 for clnt conn (was
0:af15542d90e3a9f5d14e9068be5f18)
0.068   log_pkt pkt.c:99 RX from=[::1]:4434 len=78 0xfe=Retry vers=0x4547471b
dcid=0:7775ee1c scid=0:59dbff15
tok=8393ca9cbbcc6b5d751eceefce35bd363887466023ccab95487f9de78a1a2473af15542d90
e3a9f5d14e9068be5f18 rit=77545599b9a8d707f25b5a3637ee56b7
        rx_pkt conn.c:1147 handling serv retry w/tok
8393ca9cbbcc6b5d751eceefce35bd363887466023ccab95487f9de78a1a2473af15542d90e3a9
f5d14e9068be5f18
0.069   log_pkt pkt.c:156 TX to=[::1]:4434 0xc0=Initial vers=0x4547471b
dcid=0:59dbff15 scid=0:7775ee1c
tok=8393ca9cbbcc6b5d751eceefce35bd363887466023ccab95487f9de78a1a2473af15542d90
e3a9f5d14e9068be5f18 len=0 nr=0
        enc_padding_frame frame.c:1450 PADDING len=25
        log_stream_or_crypto_frame frame.c:156 CRYPTO off=0 len=295 [seq]
        enc_padding_frame frame.c:1450 PADDING len=794
        log_cc recovery.c:143 clnt conn 0:7775ee1c: in_flight=1200 (+1200),
cwnd=12000 (+0), ssthresh=0 (+0), srtt=0.000 (+0.000), rttvar=0.000 (+0.000)
        log_sent_pkts conn.c:292 clnt Initial unacked: 0
0.071   add_dcid conn.c:807 hshk switch to dcid 0:003b3553 for clnt conn (was
0:59dbff15)
0.071   log_pkt pkt.c:109 RX from=[::1]:4434 len=43 0xc0=Initial
vers=0x4547471b dcid=0:7775ee1c scid=0:003b3553 tok= len=41 nr=0
        dec_ack_frame frame.c:565 ACK 0x02= lg=0 delay=0 (0 usec) cnt=0 rng=0
[0]
```

```
        dec_ack_frame frame.c:625 ECN verification failed for clnt conn
0:7775ee1c
        log_cc recovery.c:143 clnt conn 0:7775ee1c: in_flight=0 (-1200),
cwnd=13200 (+1200), ssthresh=0 (+0), srtt=0.003 (+0.003), rttvar=0.001
(+0.001)
        dec_close_frame frame.c:713 CONNECTION_CLOSE 0x1c=quic err=0x103
frame=0x6 rlen=14 reason=TLS error 515
        q_connect quic.c:305 clnt conn 0:7775ee1c not connected
        q_close quic.c:731 closing clnt conn 0:7775ee1c on [::1]:59284 w/err
0x0
        q_close quic.c:802 clnt conn 0:7775ee1c stats:
        q_close quic.c:804 pkts_in_valid = 3
        q_close quic.c:806 pkts_in_invalid = 0
        q_close quic.c:807 pkts_out = 3
        q_close quic.c:808 pkts_out_lost = 0
        q_close quic.c:809 pkts_out_rtx = 0
        q_close quic.c:812 rtt = 0.003 (min = 0.000, max = 0.000, var = 0.001)
        q_close quic.c:814 cwnd = 13200 (max = 13200)
        q_close quic.c:816 ssthresh = 0
        q_close quic.c:817 pto_cnt = 0
        q_close quic.c:818 frame                  code        out         in
        q_close quic.c:824 PADDING                0x00       2530          0
        q_close quic.c:824 ACK                    0x02          0          1
        q_close quic.c:824 CRYPTO                 0x06          3          0
        q_close quic.c:824 CONNECTION_CLOSE_QUIC  0x1c          0          1
        q_close quic.c:826 strm_frms_in_seq = 0
        q_close quic.c:827 strm_frms_in_ooo = 0
        q_close quic.c:828 strm_frms_in_dup = 0
        q_close quic.c:829 strm_frms_in_ign = 0
        free_stream stream.c:140 freeing strm 0 on clnt conn 0:7775ee1c
0.077   main client.c:625 client exiting
```

3.3 Métriques

Notre environnement réseau ne nous a pas permis de voir une différence significative entre l'utilisation de TCP/TLS et HTTP3. En effet, les latences dans un réseaux local sont presque nulles et un réseau métropolitain (ici l'infrastructure Orange toulousaine pour les particuliers) ne permet pas de produire des conditions contrôlées car la charge du réseau ne dépend pas de nous.

4 Conclusion

Des expérimentations futures seraient à prévoir pour certains points que l'on n'a pas pu traiter à cause de limitations. Lorsque les implémentations auront assez évolué

pour être compatibles les unes avec les autres, il pourra être intéressant de les comparer. Nous nous sommes heurtés à des problèmes de reproductibilité (build des implémentations, et environnement réseau), et au final nous n'avons pu qu'observer le contenu des échanges QUIC où l'on retrouve bien les éléments de protocole attendus (crypto data, connection stream id…), mais nous n'avons pas pu obtenir de métriques autres que le nombre d'aller-retour.